(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 665 670 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **95300247.4**

(22) Date of filing : **16.01.95**

(51) Int. Cl.^{6} : **H04L 29/06**

(30) Priority : **14.01.94 US 182969**

(43) Date of publication of application :
02.08.95 Bulletin 95/31

(84) Designated Contracting States :
DE FR GB

(71) Applicant : **TRAVELING SOFTWARE, INC.**
North Creek Corporate Center,
18702 North Creek Parkway
Bothell, WA 98011 (US)

(72) Inventor : **Pyne, Charles F.**
187 Seekonk Street
Norfolk, Massachusetts 02056 (US)

(74) Representative : **Lawrence, Malcolm Graham**
Hepworth, Lawrence, Bryer & Bizley
Merlin House
Falconry Court
Bakers Lane
Epping Essex CM16 5DQ (GB)

(54) Remote file transfer method and apparatus.

(57) Remote file transfer applications often involve a situation where a receiving computer (22) contains a reference file (48) that may be similar, or perhaps even identical to, a source file (46) to be transmitted by a sending computer (20). Disclosed is a file transfer method that identifies and isolates the differences between the two files, and transmits only those differences to the receiving computer. The method divides the data in the reference file into a plurality of blocks and associates each block of data with a key value. The key values are then sent to the sending computer in the form of an array. At the sending computer, a block of data at the source file is identified, its key value computed, and the key value is then compared to the keys in the array. If a match is found, an indication of such is sent to the receiving computer. Otherwise, a byte of data from the data block is sent to the receiving computer, and a subsequent block of data is identified and analyzed. The latter steps of the method are repeated until a representation of the source file is present at the receiving computer.

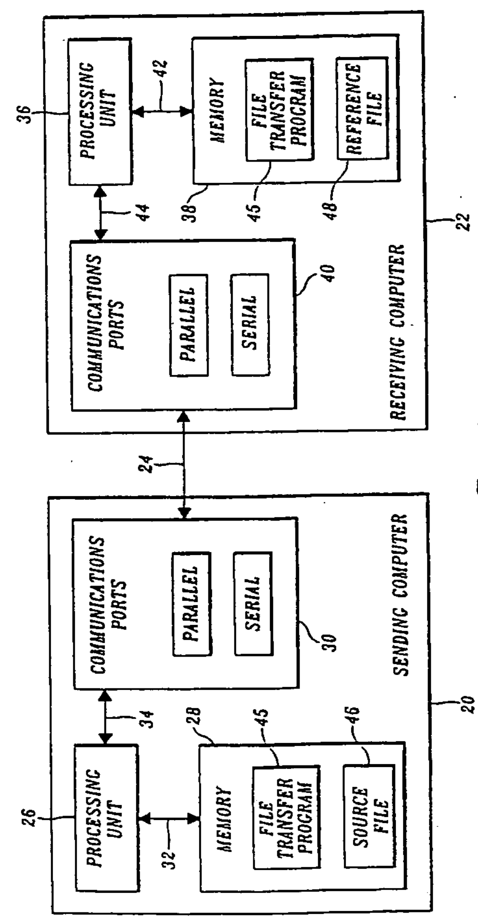

Fig. 1.

EP 0 665 670 A2

Jouve, 18, rue Saint-Denis, 75001 PARIS

The present invention relates to computer communications in general and, in particular, to a method and apparatus for decreasing the time required to update files located at a remote computer.

In computer communications technology, the rate of data communication between a computer and other peripheral devices is very important. The ability to quickly and accurately transfer data between two personal computers is of special interest in light of the increased use of portable computers. Often, data entered into a portable computer is ultimately transferred to a user's home or office personal computer. Computer specialists are continually searching for communication protocols that decrease the time required to transfer data without compromising the reliability of the data being transmitted.

A conventional method for conveying data between computers, especially personal computers, involves the interconnection of a data bus in a sending computer with a data bus in a receiving computer. This may be done by coupling the serial, parallel, or similar communications ports of each computer through an interface link, such as a cable or across a data path using modems. In serial communication, data is transferred one bit at a time. Serial communications work well for transferring data over long distances, and particularly with modems that couple two computers using a telephone line. However, the time required to transfer data using serial communications can be significant, especially for larger files. When communicating between two devices that are relatively close, parallel communications are often used. Parallel communication is the simultaneous transfer of a number of bits of data in parallel, e.g., 8-bit, using a multi-bit data path.

Computer software companies are continually investigating more efficient methods of transferring data to reduce data transmission times. Two prevalent areas of concentration have been on increasing data transfer rates and on incorporating forms of data compression to reduce the amount of data being sent. Advances in data transfer rates have been accomplished by increasing the speed at which modems communicate in serial communication and by increasing the number of bits that can be transferred simultaneously in parallel communication. An example technology that incorporates the latter technique is described in U.S. Patent No. 5,261,060, titled "Eight-bit Parallel Communications Method and Apparatus," and assigned to the assignee of the present invention. U.S. Patent No. 5,261,060 is hereby incorporated by reference. Data compression schemes reduce the size of a file to be transmitted by various means of compacting information. For example, one common compression technique, called key-word encoding, replaces words that occur frequently, e.g., *the*, with a 2-byte token representation of each word. After the compressed data is received by a remote computer, the data is decompressed to create a representation of the original contents of the file.

A more recent approach to decreasing the time required to transfer a file has recognized that a receiving computer will often have a file, i.e., a reference file, that is similar or perhaps even identical to a source file to be transmitted. For example, the source file may simply include text from the reference file with only a few words or sentences changed. Rather than sending an original or compressed representation of the entire source file, file transfer methods utilizing this approach identify the differences between the two files, and then transfer only the differences to the receiving computer. Upon receipt, the difference information is used to update the reference file at the receiving computer, thereby reproducing a precise copy of the source file. The present invention is directed toward an improved method of identifying and transferring revisions between a source file and a reference file to create an accurate copy of the source file at a remote computer.

The invention is a file transfer method that identifies and isolates the differences between a source file located at a sending computer and a reference file, located at a receiving computer, that may have data similar to the data comprising the source file. The computers are connected through a computer data interface. The method includes the steps of: (a) dividing the reference file into a plurality of data blocks and associating each data block with a key value representative of the data in each block; and (b) identifying blocks of data at the source file and using the key values to compare blocks of data from the reference file to blocks of data from the source file and, in instances where a match is found between a block of data from each file, sending an indication of the match to the receiving computer so that the block of data indicated by the match need not be transmitted to the receiving computer.

In accordance with other aspects of the invention, the step of identifying blocks of data at the source file includes the step of computing a source key for each block of data which is then compared to the key values from the reference file.

In accordance with still further aspects of the invention, an initial block of data is identified from the source file and a source key is computed from the initial block. If a match for the initial block is not found, the method includes the steps of:

(i) transmitting a byte of data from the initial block to the receiving computer; and

(ii) identifying a subsequent block of data from the source file comprising the initial block of data, less the transmitted byte, and a byte of data from the source file.

In accordance with other aspects of the invention, the method includes the step of transmitting the key values associated with data blocks in the refer-

ence file to the sending computer. Further, the key value for a block of data is computed by multiplying the bytes in the block by one or more multipliers, the value of the multiplier being dependent upon the position of a given byte in the block, and summing the results of the multiplication operations.

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIGURE 1 is a block diagram of a communications network including a sending computer and a receiving computer, each running a file transfer program that may be used to update files in accordance with the invention;

FIGURE 2 is a block diagram depicting the representation of a reference file, that may have similarities to a source file to be transmitted, using a number of keys, with each key being associated with and representative of a block of data in the reference file;

FIGURE 3 is a block diagram illustrating the selection of blocks of data at the source file on a sliding window basis;

FIGURE 4 is a flow diagram of an exemplary routine for implementing a file transfer program in accordance with the invention;

FIGURE 5 is a flow diagram of a first exemplary subroutine for determining key values for each block of data in the reference file;

FIGURE 6 is a flow diagram of an exemplary routine in accordance with the invention for determining the differences in the source and reference files and transferring those differences to the receiving computer where a destination file is created;

FIGURE 7A illustrates a second exemplary method of determining key values for each block of data in the reference file;

FIGURE 7B is a flow diagram of a subroutine for implementing the method determining key values shown in FIGURE 7A; and

FIGURE 7C is a flow diagram of a subroutine for determining the value of a key associated with a current block of data in the source file.

Remote file transfer applications often involve a situation where a receiving computer already contains a file that is similar, or perhaps even identical to, a file to be transmitted. For example, the file to be transmitted may be a revision of a text file with only a few words or sentences changed. The invention is a file transfer method that identifies and isolates the differences between the two files, and transmits only those differences to the receiving computer. For similar files, the file transfer method can result in compression ratios far in excess of those achieved by traditional data compression methods.

FIGURE 1 illustrates a typical operating environment in which the invention may be utilized. A sending computer 20 is coupled to a receiving computer 22 through a communications link 24. The computers are of a type generally known in the art, such as personal or laptop computers. The communications link may be any known means for transferring data between the two computers. such as the LAPLINK® series of file transfer tools manufactured and sold by Traveling Software. Inc., the assignee of the present invention.

The sending computer 20 generally comprises a processing unit 26, a memory 28, and a number of communications ports 30. The memory, including random access memory (RAM), read only memory (ROM), and external systems memory, is connected to the processing unit 26 by a data/address bus 32. The communications ports are connected to the processing unit by a data bus 34. The communications ports 30 include parallel and serial ports, as well as other input/output technologies including PCMCIA card technology, that allow data to be sent and received by the sending computer. The receiving computer 22 is similar to the sending computer, and includes a processing unit 36, memory 38, communications ports 40, data/address lines 42, and a data bus 44. Although for ease of description one computer is called the sending computer and the other is called the receiving computer, the computers are generally interchangeable.

In order to accomplish data transfer, the sending and the receiving computers include computer program controls that, for example, are stored in RAM and executed by the processing units of each computer. In one embodiment of the invention, the sending and receiving computer controls are combined into a single file transfer program 45 that is resident at each computer. In this manner, each computer can operate as a sending or receiving computer. Because of the requirements of handshaking, copies of the file transfer program 45 located at each computer are preferably executed simultaneously. This allows for full-duplex transmission, i.e., simultaneous communications in each direction. The invention may also be utilized in half-duplex communications, although not as efficiently.

For clarity in this discussion, throughout the Detailed Description it is assumed that a source file 46 located at the sending computer is to be sent to the receiving computer 22. Further, it is assumed that the receiving computer includes a reference file 48 that includes at least some similarities to the source file. Once a user indicates that a source file is to be transferred, a reference file that may have data that is similar to the source file is identified by, for example, having a file name that is the same or similar to the source file. The invention described herein generally assumes that a reference file has been identified.

The basic steps implemented by the file transfer program are as follows:

(1) identifying a reference file at the receiving computer that may have data similar to data comprising the source file;
(2) dividing the data comprising the reference file into a plurality of data blocks having n-bytes per block and associating each data block with a key value;
(3) transmitting the key values from the receiving computer to the sending computer;
(4) identifying a current n-byte block of data from the source file and computing a value for a source key associated with the current block of data;
(5) comparing the value of the source key with each of the key values from the reference file and, if a match is found, (i) transmitting an indication of the match to the receiving computer, and (ii) repeating step (4); and
(6) if a match was not found, transferring to the receiving computer a byte of data from the current block of data, adding an additional byte of data from the source file to the current block of data, re-computing the value of the source key, and repeating step (5).

Generally, the loops created by steps (5) and (6) repeat until all of the data in the source file has been considered. At the receiving computer, a destination file is created from the match indications and the byte transmissions. The destination file will be a duplicate of the source file upon completion of transmission.

FIGURE 2 illustrates pictorially step (2), which includes dividing the data comprising the reference file into a plurality of data blocks 50_a, 50_b, 50_c...50_y, 50_z and associating each data block with a key value 52_a, 52_b, 52_c ... 52_y, 52_z . It is noted that the last block of data may include less than n-bytes, and thus is indicated as having x-bytes. Once the reference file is separated into the data blocks 50, the key value 52 of each block may be computed using a number of methods. In a first exemplary embodiment, each key is computed by adding the value of each byte of data in the block to produce a total of all of the bytes in the block. By way of background, each 8-bit character in any given block is representative of an ASCII value that ranges from 0 to 255, i.e., $2^8 - 1$. ASCII is an acronym for the American Standard Code for Information Interchange, a coding scheme that assigns numeric values to letters, numbers, punctuation marks, and certain other characters. Through the standardization of values used for such characters, ASCII enables computers and computer programs to exchange information. Calculating the keys as described above will produce a whole number between zero and n^2 that is representative of the data contained in any given block.

Once the keys for each block of data in the reference file have been computed, the keys are sent as an array to the sending computer for comparison to the source file.

FIGURE 3 is a pictorial representation of steps (3)-(5). In step (3), a current n-byte block of data from the source file is identified and a value for a source key associated with the current block of data is computed. Thus, in the first comparison bytes zero through (n-1) are identified as the current block of data. Thereafter, the key value of the current block of data is computed using the same method that was used to compute the keys in the reference file. The key value for the current block of data is labeled KEY1.

In step (4), the value of KEY1 is compared to each of the keys in the reference file to determine whether a match has been found, thereby indicating that the current block of data is identical to a block of data in the reference file. If a match is found, an indication of such is sent to the receiving computer. Assuming a match has not been found, according to step (5) the first byte in the current block (byte zero) is sent to the receiving computer. A subsequent "current" block of data is then evaluated by subtracting the first byte of data (byte zero) from the current block, adding the next sequential byte of data (byte n) to the current block, and re-computing the key value for the subsequent current block. The key value for this block of data is labeled KEY2. Thus, KEY2 will comprise the values of bytes 1 through n. The value of KEY2 is then compared to each of the keys in the key array for the reference file.

Assuming a match is not made, the first byte in the current block (byte 1) is sent to the receiving computer. A third key KEY3 representing the current block of data is then computed by sliding the current block of data one byte to the right, such that KEY3 comprises the values of bytes 2 through (n+1). This will continue until either a match is found between a key value computed from a data block in the source file and a key in the key array for the reference file, or all of the data in the source file has been transmitted. Assuming a match is found, an indication of such is sent to the receiving computer, and a subsequent current block of data is computed from the source file.

It is noted that in the case where a match is not found, the additional time required to transfer a file, in comparison to traditional methods, is negligible despite the sliding window and key computations. This is, in part, due to the fact that the processor can make computations much faster than data can be sent. Further, in a preferred embodiment, the receiving computer is configured to expect that bytes being received are data bytes and are *not* indicative of a match between two data blocks. In the latter case, an additional "match-indicator" byte is sent ahead of the byte(s) indicating that a match has occurred. Thus, the number of bytes being sent in the case of no matches will generally be approximately the same as

if the data were simply transmitted without any opportunity for match checks in accordance with the invention.

The foregoing is an overview of an exemplary embodiment of the file transfer program 45. Exemplary routines for implementing the file transfer program in software are set forth in FIGURES 4-6 and accompanying text. In that regard, FIGURE 4 is a flow diagram of a routine for computing a key array from the contents of the reference file. The size of each block of data is set at block 100. In one embodiment, each block contains 256 bytes. At block 102, the variable nBlock, representing the current block of data being considered by the routine, is set to zero. The value of the key for the current block of data is computed at block 104. A suitable routine for computing the key value is illustrated in FIGURE 5. At block 106, the array BlockKey[nBlock] is set equal to the value of the key computed for the current block. The variable nBlock is then incremented at block 108. A test is made at block 110 to determine whether the end of the reference file has been reached. If the end of the file has not been reached, the routine loops back to block 104. If the end of the file has been reached, the BlockKey array is sent to the sending computer at block 112 and the routine terminates.

FIGURE 5 is a flow diagram of a first exemplary subroutine suitable for use in FIGURE 4 (block 104) for computing the value of the key associated with a given block of data. The subroutine will be called for each data block in the reference file. At block 120, the variable "n," which is representative of the byte count, is set equal to zero. At block 122, the variable "key" is also set equal to zero. A byte of data is then read from the reference file at block 124. Variable n is incremented at block 126. At block 128, the key is set equal to its previous value plus the value of the current byte of data that was read at block 124.

A test is made at block 130 to determine whether the end of the reference file has been reached. If the end of the reference file has not been reached, a test is made at block 132 to determine whether a full block of data has been considered, i.e., whether n is equal to the block size. If n is not equal to the block size, the subroutine loops back to block 124. If n is equal to the block size, or if it was determined at block 130 that the end of the file was reached, the subroutine terminates, and control returns to the routine of FIGURE 4.

FIGURE 6 is a flow diagram of a routine for comparing keys associated with n-byte blocks of data from the source file with the keys computed from the reference file and contained in the BlockKey array. At block 150, the variable "current key" is set equal to zero. A test is made at block 152 to determine whether there is at least n-bytes of data in the source file that have yet to be compared. If there are at least n-bytes of data not yet compared, at block 154 an n-byte block of data is read from the source file. At block 156, the value of the current key, representing the current block of data, is computed using the same computation methods that were used in FIGURE 5, i.e., by adding the weighted value of each byte in the current block of data. The key values in the BlockKey array are then searched at block 158 to determine whether any of the keys in the BlockKey array match the current key. The test for whether a match is found is performed at block 160.

If a match was found at block 160, a message is sent at block 162 to the receiving computer to emit the matching block to the destination file. The routine then loops to block 154. If a match was not found, at block 164 a first byte of data in the current block is sent to the receiving computer. At block 166, the byte of data that was sent to the receiving computer is removed from the current block of data. A test is then made at block 168 to determine whether there is any data remaining in the source file that has not been considered. If there is data remaining in the source file, a new byte of data is read from the source file at block 170 and added to the current block of data at block 172. The routine then loops to block 156 where the key for the current block of data is computed.

Those skilled in the art will appreciate that, in computing the current key in block 156 after looping from block 172, it is more efficient to obtain the value of the key for the current block by subtracting from the previously-computed current key the value of the byte that was removed from the current block (in block 166) and then adding the value of the byte that was added to the current block (in block 172), rather than performing the key calculation by adding every character in the current block.

If all of the data remaining in the source file has been considered, or if there were less than n-bytes of data remaining in the source file as determined in the test at block 152, the data remaining in the source file is sent to the receiving computer at block 174. At the receiving computer, the transmitted data is added to the destination file and the file transfer is complete. In an alternative embodiment, instead of simply sending the data remaining as indicated in block 174, a test is made to determine if the key value of the data remaining matches the key value of the last block of data in the reference file. If a match is present, an indication of the match is sent to the receiving computer, and the data itself need not be transmitted. Otherwise, the actual data is transmitted. This will result in a further optimization of the transfer in situations where the end of the reference file contains the same data as the end of the source file.

Once all of the data in the source file has been added to the destination file, i.e., through blocks 162, 164 and 174, the destination file will in most circumstances be an exact copy of the source file. However, it is preferable that a check be made to ensure that the destination file is indeed a precise duplicate of the

source file. In block 176, the integrity of the destination file is checked using means known to those skilled in the art. One method of checking the file integrity is a cyclic redundancy-check (CRC), such as that set forth in M. Nelson, The Data Compression Book, 446-448 (M&T Books 1991), which is hereby incorporated by reference. If the integrity of the destination file was compromised, the data from the source file is retransmitted to the destination file using conventional transmission methods. This is indicated at block 178. If the integrity of the destination file tested positive, or upon transmitting the source file, the routine terminates.

One circumstance where the destination file may not be an accurate copy of the source file is where two or more different blocks of data yield the same key value. If it is assumed that each block of data is 256 bytes, under the key computation method described in FIGURE 5, the range of possible key values is 0 to 65,280, the latter value occurring only if each byte in the block has a numerical value of 255. The odds of having duplicate keys are significantly decreased if: (1) the range of possible key values is relatively large, and/or (2) the likelihood that key computations will fall within a broader portion of the range is increased. In light of the above, it will be appreciated that the accuracy of the data file transfer method for transmitting data in accordance with the invention will work most effectively if the possibility of having two different blocks of data having the same key value is extremely remote.

Another desirable feature of an advantageous key computation method is if the current key values for blocks of source data that are derived on a sliding window basis can be quickly established. One way of accomplishing this is to have a key computation method that allows the current key to be updated by subtracting the key value associated with the byte of data to be subtracted from the current block of data (block 166) and adding the key value associated with the byte of data to be added to the current block (block 172). While the key computation method described in FIGURE 5 has this desirable feature, it may not work well for larger files because of its limited range of possible key values and the distribution of values within this range.

FIGURES 7A-7C illustrate a second exemplary embodiment for calculating keys in accordance with the invention in which the range of possible key values is extended beyond the summing scheme of FIGURE 5, thereby decreasing the likelihood that any key value will be representative of more than a single block of data. Further, the calculation method allows the current key to be updated very quickly, as described in FIGURE 7C and accompanying text. The examples of FIGURES 7A-7C illustrate a 32-bit key, but it will be appreciated that other key sizes may be implemented.

With reference to FIGURE 7A, the 32-bit key is divided into a lower 24-bit segment and an upper 8-bit segment. The 24-bit segment is computed using the following equation:

$$C_1(n) + C_2(n-1) + C_3(n-2) + \ldots C_{n-1}(2) + C_n \qquad (1)$$

where C_i is the character in the ith position of a current block and n is the number of bytes in each block. The upper 8-bits of the 32-bit key are calculated by performing an exclusive OR operation (XOR) on each of the characters, as shown by the equation:

$$C_1 \text{ XOR } C_2 \text{ XOR } C_3 \ldots C_{n-1} \text{ XOR } C_n \qquad (2)$$

Once the lower and upper key values are calculated, the bits are concatenated to form each 32-bit key.

FIGURE 7B illustrates a suitable subroutine for implementing the key calculations illustrated in FIGURE 7A. The subroutine is called by the routine of FIGURE 4 in lieu of calling the subroutine of FIGURE 5. At block 200, the variable "n," which is indicative of the byte count for any given block, is set equal to zero. At block 202, the lower and upper portions of the key, i.e., key.24 and key.8, are set equal to zero. The variable "sum" is set equal to zero at block 204. At block 206, a byte of data is read from the reference file.

At block 208, the value of the current byte is added to the variable sum. The key.24 variable is then increased by the value of sum at block 210. It will be appreciated that blocks 208 and 210 are alternate methods of computing equation (1) without requiring multiplication operations. At block 212, the variable key.8 is set equal to the previous value of key.8 XOR the current byte.

A test is made at block 214 to determine if the end of the reference file has been reached. If the end of the reference file has not been reached, a test is made at block 216 to determine whether n is equal to the block size. If n is not equal to the block size, the routine loops to block 206. If n is equal to the block size, or if the end of the file has been reached, the variable "key" is set by concatenating the lower 24-bit (key.24) value computed at block 210 with the upper 8-bit (key.8) value computed at block 212. The subroutine then terminates, and the program returns to block 106 of FIGURE 4.

FIGURE 7C illustrates a suitable subroutine that may be called from block 156 of FIGURE 6 to compute the value of the current key for a current block of data. The subroutine illustrates the optimization of key calculations for those blocks of source data that are identified on a sliding window basis, and thus have key values similar to a key that has already been computed. A test is made at block 220 to determine if the current block was identified on a sliding window basis, or in other words, if the subroutine was called because a match with the previous block was not found. If the current block of data was not identified on this basis, the subroutine of FIGURE 7B to com-

pute the value of the current key. This occurs at the first block of data in the source file or after a match was found.

If the current block of data was identified on the basis of a previous, unmatched block of data, the first byte from the previous block (termed the removed byte from the operation of block 166) is subtracted from sum at block 224. At block 226, the key.24 is set equal to its previous value less the product of the block size times the value of the removed byte. At block 228, the new byte (added to the current block in block 172) is added to sum. The key.24 variable is then increased by the value of sum at block 230.

At block 232, the variable key.8 is set equal to the previous value of key.8 XOR the removed byte. An exclusive OR operation is then performed between key.8 the new byte. At block 236 the variable "key" is set by concatenating the lower 24-bit (key.24) value computed at block 230 with the upper 8-bit (key.8) value computed at block 234. The subroutine then terminates, and control returns to block 158 of FIGURE 6. As can be seen, the subroutine of FIGURE 7C allows key values to be quickly computed, thus allowing faster operation of the file transfer program when looking for matches between blocks from the source file and blocks from the reference file.

As will be appreciated by those skilled in the art, a large number of different key computation methods may be used in accordance with the invention. Thus, the invention is not to be limited by exemplary key calculations illustrated herein. Any key computation that is not unnecessarily time-consuming computationally and that provides a relatively wide range of results may be beneficial. Moreover, the type of key computation used in any particular embodiment may depend upon the block size to achieve optimal results. Another key computation that may be used is to multiply each character in a block by its position in the block, and summing the results of the multiplication operations. Another key computation that may be implemented is the CRC file integrity check discussed above. Although this method is extremely accurate, it may be too slow for many applications.

With reference again to FIGURE 6, it is noted that further optimization may be achieved when searching the BlockKey array (block 158) by utilizing a binary search. A binary search is a type of search in which an item that may be present within an ordered list is found by repeatedly dividing the ordered list into two equal parts and searching the half that may contain the item. Because a binary search requires the searching list to be in a known sequence, e.g, ascending order, the BlockKey array would need to be arranged accordingly in order for the search to be effective. A suitable standard binary search is set forth in H. Schildt, The Complete C Reference, 487-488 (Osbourn McGraw-Hill 1987), which is hereby incorporated by reference.

It will be appreciated that in its method aspects, the invention provides a method of transmitting data from a source file located at a sending computer to a destination file located at a receiving computer, the computers being connected through a computer data interface, the method comprising the steps of:-

(a) identifying a reference file at the receiving computer that may have data similar to the data comprising the source file;

(b) dividing the data comprising the reference file into a plurality of data blocks having n-bytes per block and associating each data block with a reference key value determined by a key defining method;

(c) identifying an n-byte block of data from the source file and computing using the key defining method a current value for a source key associated with the identified block of data;

(d) comparing the current value of the source key with each of the reference key values and, if a match is found, (i) transferring an indication of such to the receiving computer, and (ii) repeating step (c);

(e) if a match was not found in step (d), transferring to the receiving computer a subset of the n-byte block of data, removing the subset from the n-byte block of data, adding additional data from the source file to the n-byte block of data, re-computing using the key defining method a current value of the source key, and repeating step (d).

Similarly, it would be appreciated that the invention provides a method of synchronizing data at a receiving unit with data at a source unit, comprising:-

(a) determining multiple reference keys corresponding to groups of data stored at the receiving unit;

(b) transmitting the multiple reference keys to the source unit;

(c) determining a source key corresponding to a group of source data in the source unit;

(d) comparing the source key with the multiple reference keys;

(e) transmitting data from the source unit to the receiving unit if the source key does not match any of the reference keys;

(f) transmitting a control signal from the source unit to the receiving unit if the source key matches a reference key, the control signal causing the receiving unit to use data at the receiving unit corresponding to the matched reference key; and

(g) repeating steps (c), (d), (e) and (f) for additional groups of source data in the source unit until the data at the receiving unit has been synchronized with the data at the source unit.

Claims

1. A method of transmitting data from a source file (46) located at a sending computer (20) to a receiving computer (22) , the receiving computer having a reference file (48) that may have data similar to data comprising the source file, the computers being connected through a computer data interface (24), the method comprising the steps of:

   (a) dividing the reference file into a plurality of data blocks (50a....50z), each data block having a length of n bytes, and associating each data block with a reference key value (52a....52z) determined in accordance with a key defining method by the data in that block; and

   (b) identifying blocks of data of length n bytes from the source file, determining source key values (KEY 1, KEY 2, KEY 3) in accordance with the key defining method, and using the source and reference key values to compare (158, 160) blocks of data from the reference file to blocks of data from the source file and, in instances where a match is found between a block data from each file, sending (162) an indication of the match to the receiving computer so that the block of data indicated by the match need not be transmitted to the receiving computer.

2. A method as claimed in Claim 1 and including the step of transmitting (112) the reference key values (52a....52z) associated with data blocks in the reference file (50a....50z) to the sending computer (20).

3. A method as claimed in Claim 1 or Claim 2 wherein an initial block of data is identified from the source file, a source key value is determined (156) from the initial block, and if a match for the initial block is not found, carrying out the following steps, namely:-

   (i) transmitting (164) a subset of the initial block to the receiving computer; and

   (ii) identifying from the source file a subsequent block of data of length n bytes comprising the initial block of data, less the transmitted subset, and including additional data from the source file (166, 168, 170, 172).

4. A method as claimed in any one of Claims 1 to 3 wherein the source key value (KEY 1) from the initial block is compared with each reference key value (52a....52z) in turn, and if a match is not found, a subset of the initial block of data is transferred (164) to the receiving computer (22), that subset is removed (166) from the initial block of data, additional data from the source file is added (170, 172) to the remaining block of data, a current value of the source key is re-computed (15, 6) according to the key defining method, and the re-computed source key value is compared with each reference key value in turn.

5. A method as claimed in Claim 4 wherein the recomputing step is repeated until all data in the source file has been compared (168).

6. A method as claimed in any preceding claim wherein at least a portion of the reference key value for a block of data is computed by adding the value of each byte of data in the block to produce (128) a total of all of the bytes in the block.

7. A method as claimed in Claim 6 and including step of multiplying (126, 128), before said addition, bytes in said block by one or more multipliers, the value of the multiplier being dependent upon the position of a given byte in the block.

8. A method as claimed in Claim 1 wherein the source key and the reference key include multiple bits and in which some of the bits are determined by a summing operation and some of the bits are determined by a logical operation.

9. A method as claimed in Claim 8 wherein the summing operation includes multiplying by constant coefficients (n, n-1....) the values (C_1....Cn) represented by bytes of the blocks of source data and in which the logical operation comprises an exclusive OR operation.

10. A method as claimed in Claim 9 wherein the key defining method for the blocks of data includes the following calculation:-

$$C_1(n) + C_2(n-1) + C_3(n-2) + \ldots + C_{n-1}(2) + C_n \quad (1)$$

wherein C_1 is the character in the ith position of the block of data.

11. A method as claimed in Claim 9 wherein the key defining method includes the following logical operation:-

$$C_1 \text{ XOR } C_2 \text{ XOR } C_3 \ldots C_{n-1} \text{ XOR } C_n.$$

12. A method as claimed in Claim 9 and including the step of determining a source key for the subsequent block of data by subtracting (226) C_1 (n) from the source key for the initial block and adding (228) the sum of the n bytes in the subsequent block of data, determining a source key for the subsequent block of data by performing on the source key for the initial block of data an exclusive OR operation with the transmitted subset

(232) and with the additional data (234), and concatenating (236) the results for said subsequent block of data.

13. An apparatus for synchronizing data at a source unit (20) and a receiving unit (22), the apparatus comprising means (45) for determining an array of reference keys corresponding to groups of data stored at the receiving unit; data transfer means (24) for transmitting the multiple reference keys to the source unit; means (45) for determining source keys corresponding to groups of source data in the source unit; means (45) for comparing the source keys with the multiple reference keys; means (24) for transmitting data from the source unit to the receiving unit when a source key does not match any of the reference keys; and means (24) for transmitting a control signal from the source unit to the receiving unit when a source key matches a reference key, the control signal causing the receiving unit to use a group of data at the receiving unit corresponding to the matched reference key.

14. A apparatus as claimed in Claim 13 wherein the means (45) for determining source keys determines a new source key after the means for comparing source keys has compared the previously determined source key, and wherein the means for determining source keys determines the new source key from a group of source data, the composition of which is determined by whether the previously compared source key matched a reference key.

15. A apparatus as claimed in Claim 13 or Claim 14 wherein the means for transmitting data (24) from the source unit to the receiving unit when a source key does not match any of the reference keys transmits less data than is included in the groups of source data used by the means for determining the source keys.

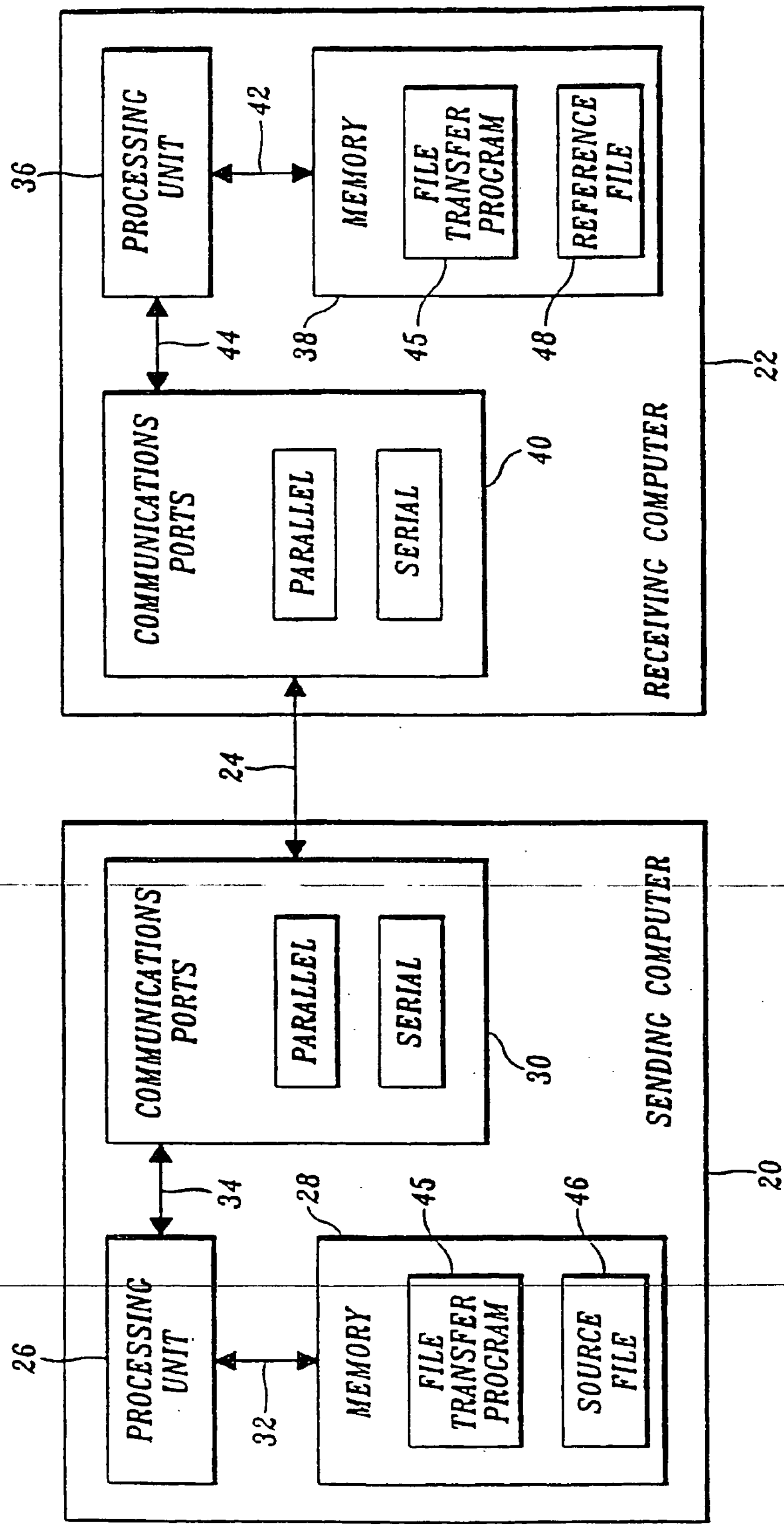
PROCESSING UNIT
36
42
MEMORY
FILE TRANSFER PROGRAM
REFERENCE FILE
44
38
45
48
22
COMMUNICATIONS PORTS
PARALLEL
SERIAL
40
RECEIVING COMPUTER
24
COMMUNICATIONS PORTS
PARALLEL
SERIAL
30
SENDING COMPUTER
34
28
45
46
20
PROCESSING UNIT
26
32
MEMORY
FILE TRANSFER PROGRAM
SOURCE FILE

Fig. 1.

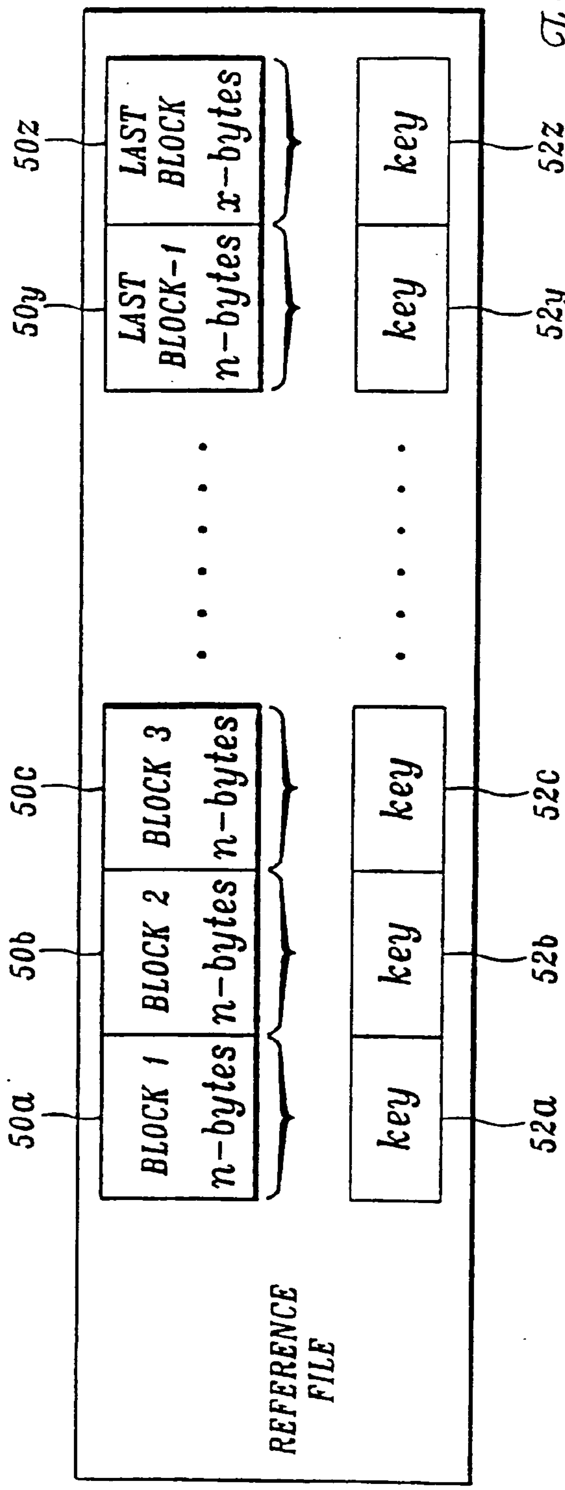
50a
50b
50c
50y
50z
BLOCK 1 n-bytes
BLOCK 2 n-bytes
BLOCK 3 n-bytes
LAST BLOCK-1 n-bytes
LAST BLOCK x-bytes
key
key
key
key
key
52a
52b
52c
52y
52z
REFERENCE FILE

Fig.2.

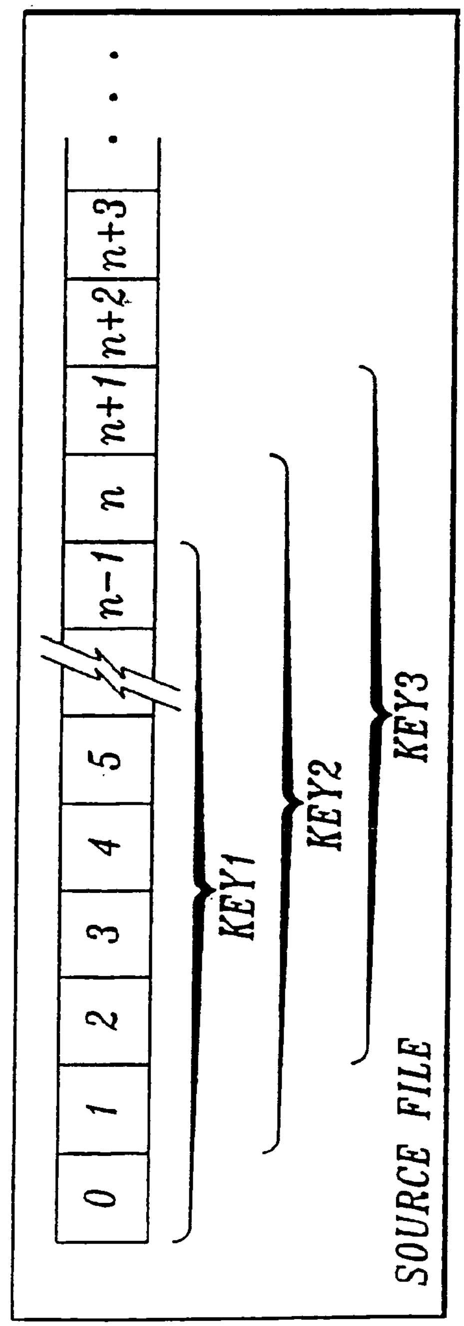
0
1
2
3
4
5
n-1
n
n+1
n+2
n+3
KEY1
KEY2
KEY3
SOURCE FILE

Fig.3.

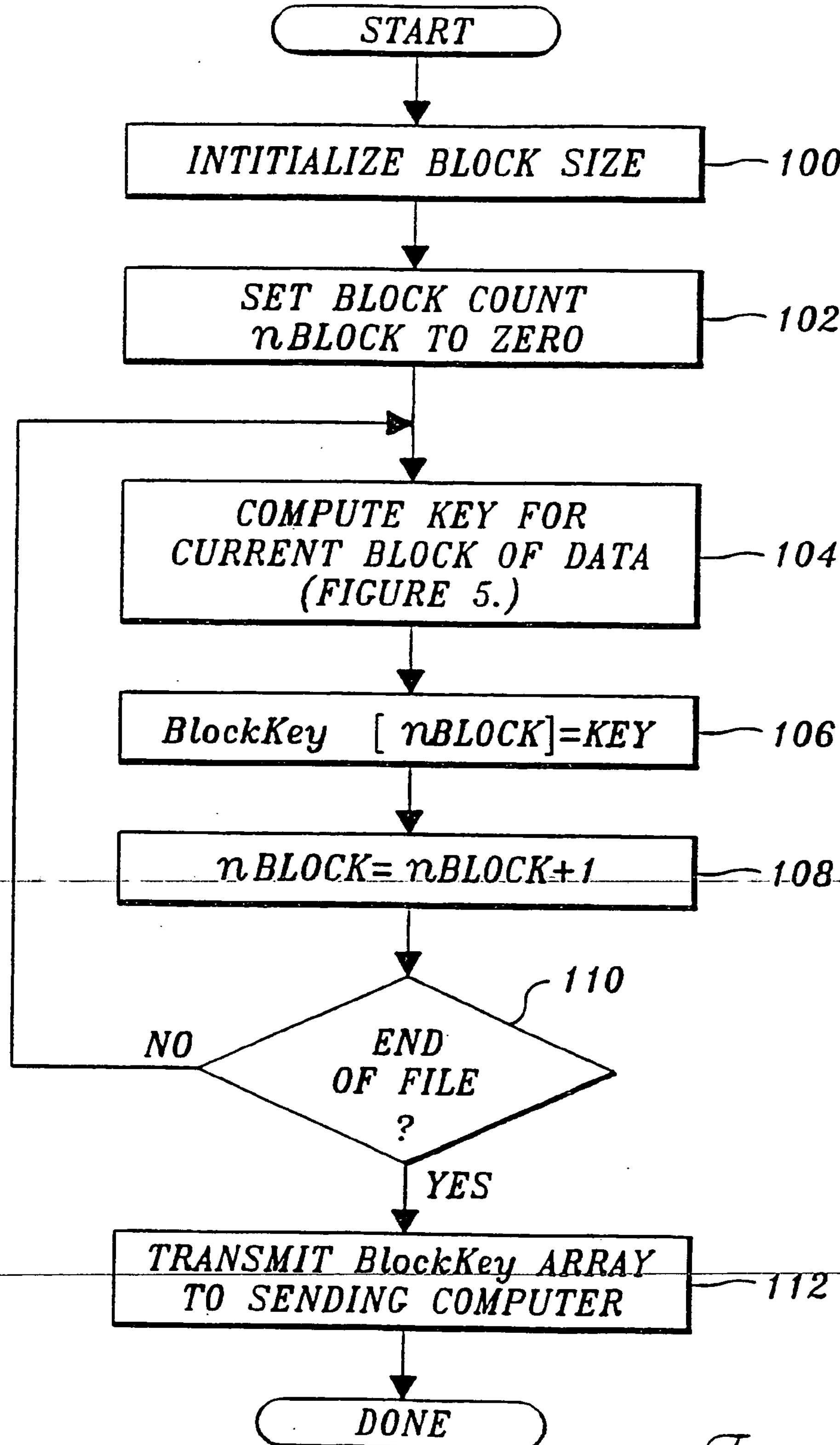
START
INTITIALIZE BLOCK SIZE
100
SET BLOCK COUNT
nBLOCK TO ZERO
102
COMPUTE KEY FOR
CURRENT BLOCK OF DATA
(FIGURE 5.)
104
BlockKey [nBLOCK]=KEY
106
nBLOCK=nBLOCK+1
108
110
END
OF FILE
?
NO
YES
TRANSMIT BlockKey ARRAY
TO SENDING COMPUTER
112
DONE

Fig.4.

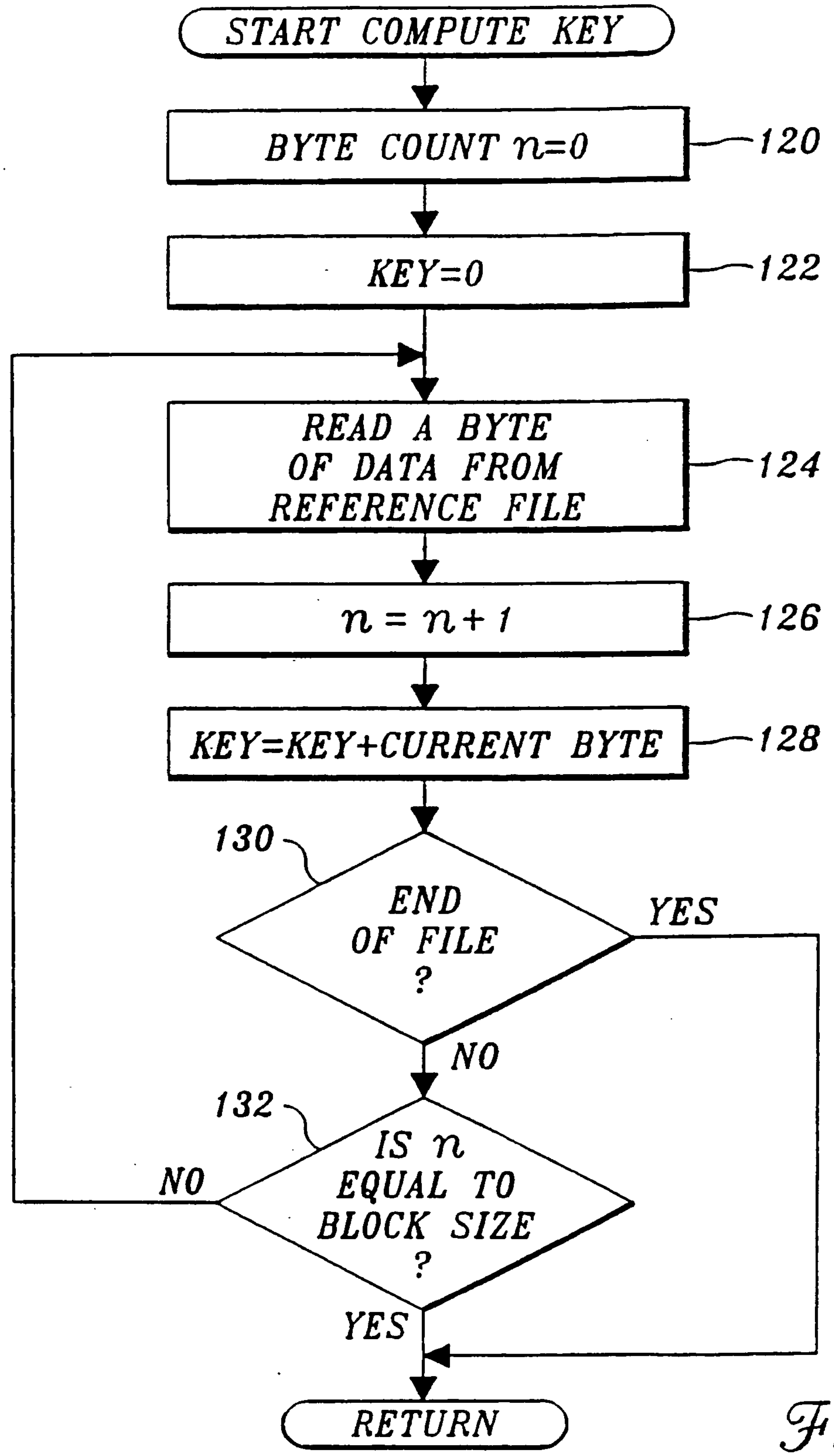
START COMPUTE KEY
BYTE COUNT n=0
120
KEY=0
122
READ A BYTE OF DATA FROM REFERENCE FILE
124
n = n+1
126
KEY=KEY+CURRENT BYTE
128
130
END OF FILE ?
YES
NO
132
IS n EQUAL TO BLOCK SIZE ?
NO
YES
RETURN

Fig.5.

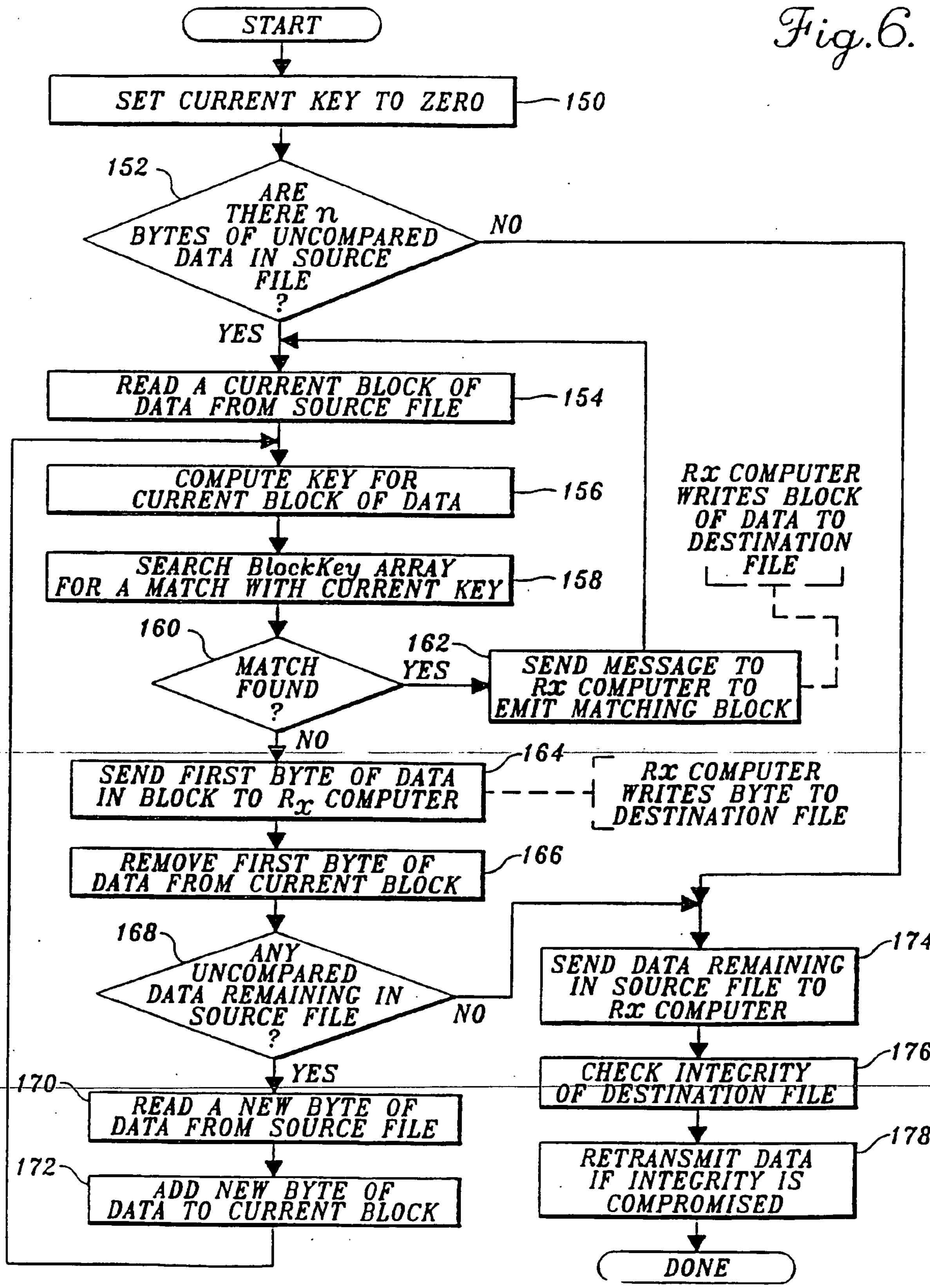
START
SET CURRENT KEY TO ZERO
150
152
ARE THERE n BYTES OF UNCOMPARED DATA IN SOURCE FILE ?
NO
YES
READ A CURRENT BLOCK OF DATA FROM SOURCE FILE
154
COMPUTE KEY FOR CURRENT BLOCK OF DATA
156
SEARCH BlockKey ARRAY FOR A MATCH WITH CURRENT KEY
158
Rx COMPUTER WRITES BLOCK OF DATA TO DESTINATION FILE
160
MATCH FOUND ?
162
YES
SEND MESSAGE TO Rx COMPUTER TO EMIT MATCHING BLOCK
NO
SEND FIRST BYTE OF DATA IN BLOCK TO Rx COMPUTER
164
Rx COMPUTER WRITES BYTE TO DESTINATION FILE
REMOVE FIRST BYTE OF DATA FROM CURRENT BLOCK
166
168
ANY UNCOMPARED DATA REMAINING IN SOURCE FILE ?
NO
YES
SEND DATA REMAINING IN SOURCE FILE TO Rx COMPUTER
174
170
READ A NEW BYTE OF DATA FROM SOURCE FILE
CHECK INTEGRITY OF DESTINATION FILE
176
172
ADD NEW BYTE OF DATA TO CURRENT BLOCK
RETRANSMIT DATA IF INTEGRITY IS COMPROMISED
178
DONE

Fig.6.

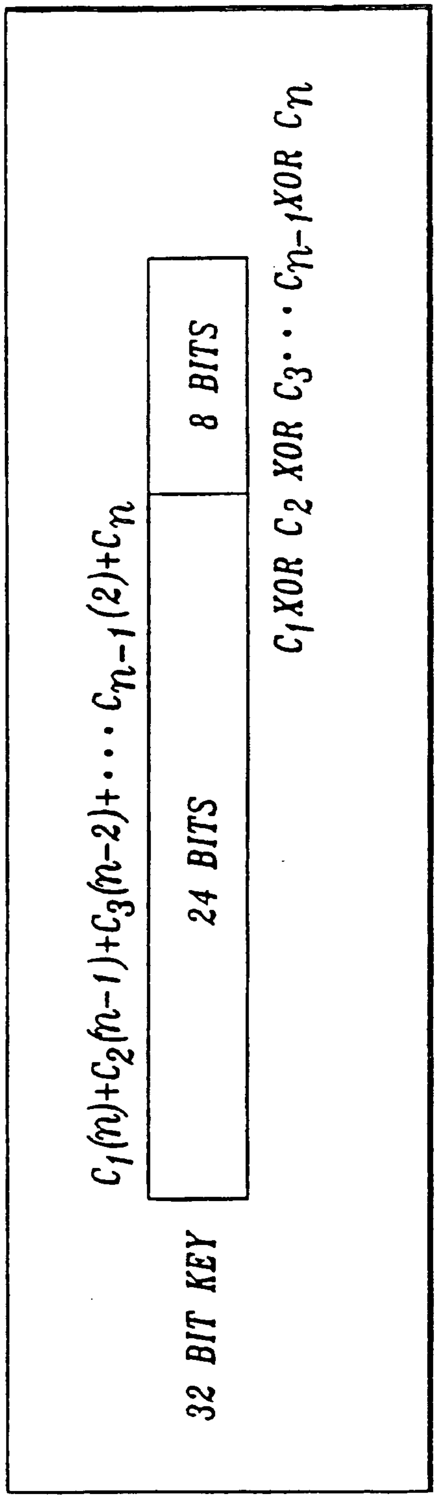
32 BIT KEY
$C_1(n)+C_2(n-1)+C_3(n-2)+\cdots C_{n-1}(2)+C_n$
24 BITS
8 BITS
$C_1 \text{ XOR } C_2 \text{ XOR } C_3 \cdots C_{n-1} \text{ XOR } C_n$

Fig. 7A.

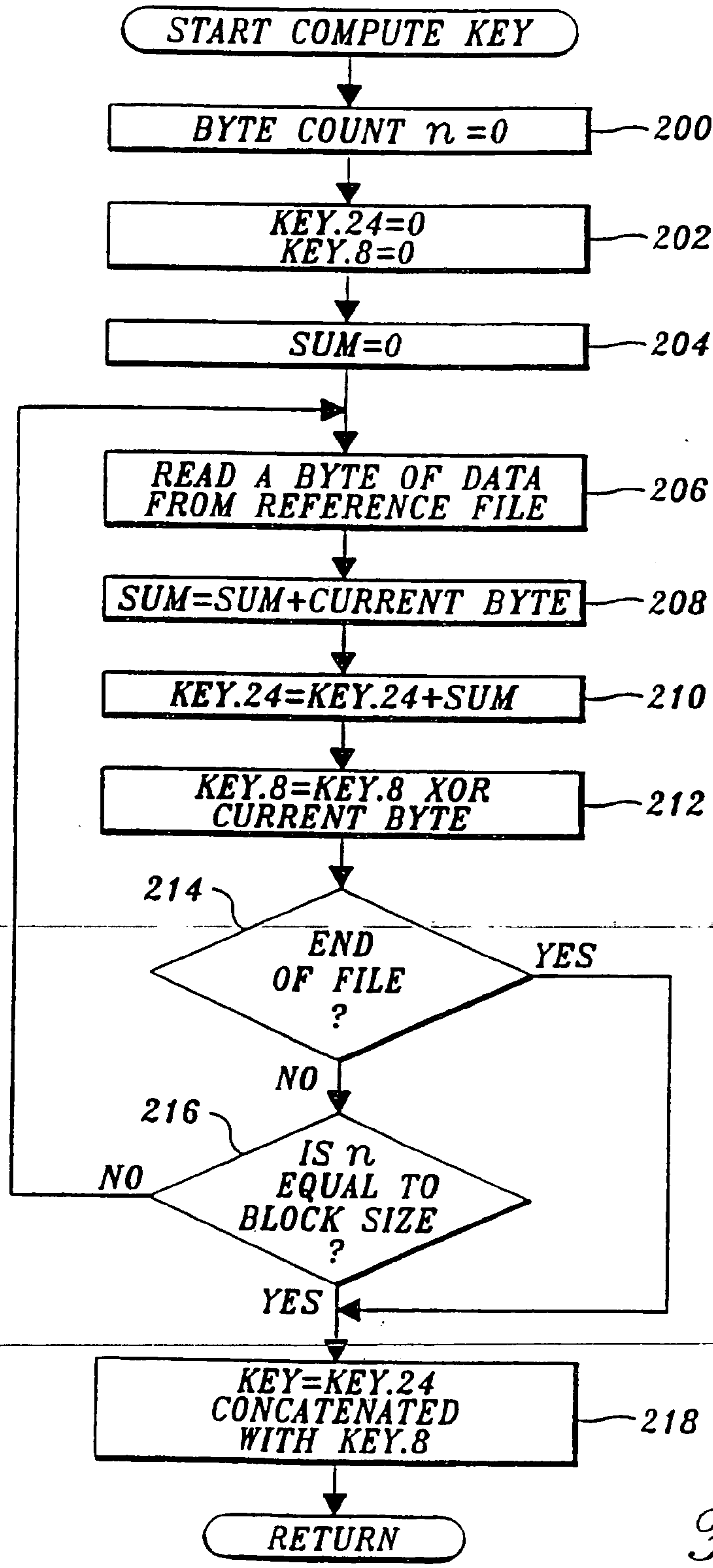
START COMPUTE KEY
BYTE COUNT n=0
200
KEY.24=0
KEY.8=0
202
SUM=0
204
READ A BYTE OF DATA FROM REFERENCE FILE
206
SUM=SUM+CURRENT BYTE
208
KEY.24=KEY.24+SUM
210
KEY.8=KEY.8 XOR CURRENT BYTE
212
214
END OF FILE ?
YES
NO
216
IS n EQUAL TO BLOCK SIZE ?
NO
YES
KEY=KEY.24 CONCATENATED WITH KEY.8
218
RETURN

Fig. 7B.

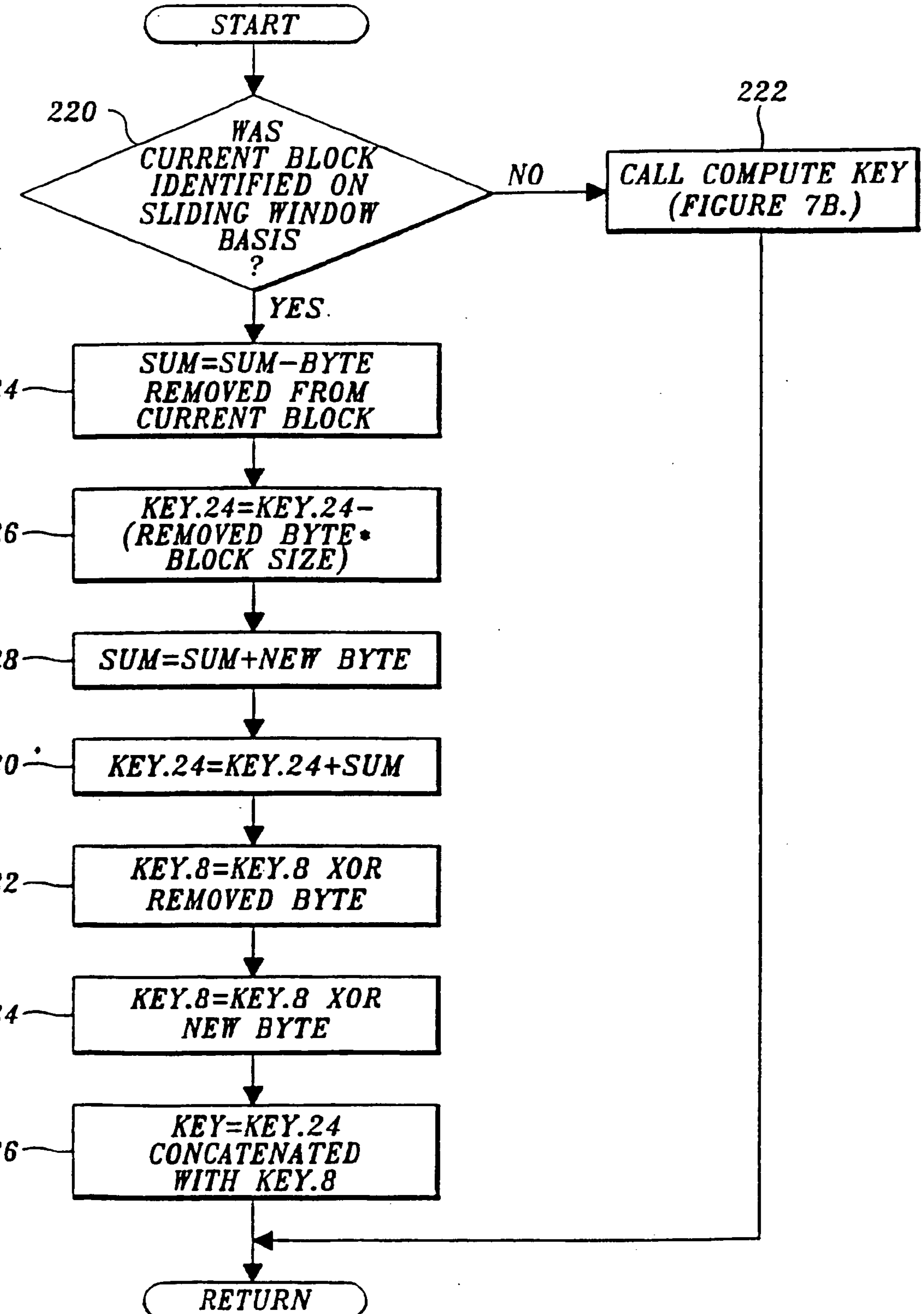
START
220
WAS CURRENT BLOCK IDENTIFIED ON SLIDING WINDOW BASIS ?
NO
222
CALL COMPUTE KEY (FIGURE 7B.)
YES
224
SUM=SUM-BYTE REMOVED FROM CURRENT BLOCK
226
KEY.24=KEY.24-(REMOVED BYTE* BLOCK SIZE)
228
SUM=SUM+NEW BYTE
230
KEY.24=KEY.24+SUM
232
KEY.8=KEY.8 XOR REMOVED BYTE
234
KEY.8=KEY.8 XOR NEW BYTE
236
KEY=KEY.24 CONCATENATED WITH KEY.8
RETURN

Fig. 7C.